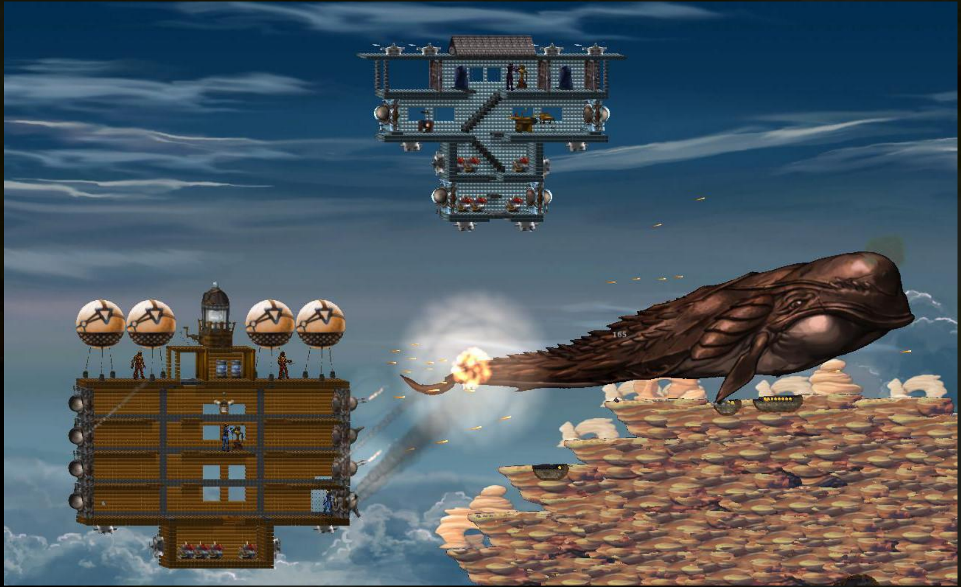
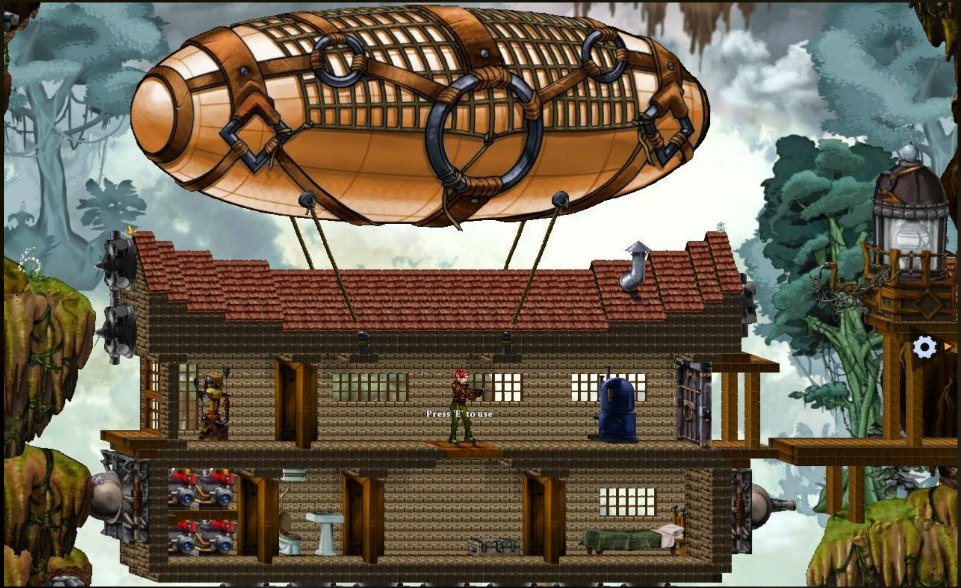
Applied Physics in Windforge

Random Windforge Facts



- 2D action building block RPG
- Large 2D procedural world
- World is destructible and can be modified by the player

Random Windforge Facts



- Buildable airships with functional parts
- 2D platform shooter
- Sky planet with floating islands and dieselpunk technology

Random Windforge Facts

- Custom in-house game engine.
- OpenGL
- Using a modified version of Box2D for physics

Windforge Physics In Action

Main Topics Covered

- Extending Box2D functionality
- Selected platforming physics problems
- Ship physics overview
- Grappling hooks

Extending Box2D

Disclaimer

- We used a version of Box2D from early 2012

- Some of the Box2D info in this talk might be out of date now.

Windforge Physics Requirements

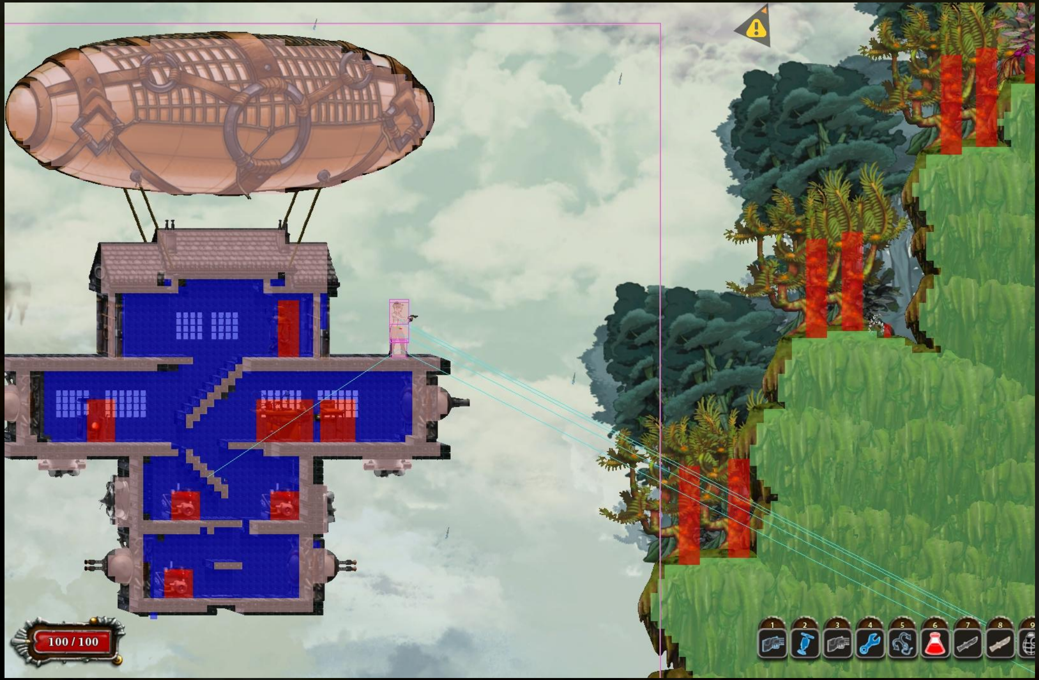
- Lots of objects interacting efficiently
- Responsive arcade platforming action
- One-way platforms and stairs
- Create and destroy large amounts of blocks easily and efficiently
- Cheap and easy to use
- (and more?)

Creating and Destroying Large Blocks Efficiently

- With the exception of this, Box2D was able to meet all of our requirements without modifications

- A few things we tried before modifying Box2D:
  - Making each block a physics body with a box collision, and connecting them all using joints
  - Making every disjoint moveable block object a physics body, and using a separate shape for each block.

Creating and Destroying Large Blocks Efficiently



- Our solution was to add a new type of shape to Box2D.

- We called it a *RasterShape*

How to Add a New Shape to Box2D

- Extend b2Shape class and implement these functions:
  - TestPoint
  - RayCast
  - ComputeAABB
  - ComputeMass

- Make b2Contact objects for the shape collision pairs you want to support
  - Ex: Polygon-Raster, Raster-Raster, Raster-Circle, etc.
  - You will also need to implement collision detection code for these combinations

- Debug drawing

Box2D Shape Requirements

- Without extra modifications Box2D shapes need to be:
    - Convex
    - Non-deformable

- Unfortunately our raster shapes were both non-convex and deformable

Fixing RasterShape Non-Convex Problems

- Added special case code in a few spots for raster collisions that can detect and add multiple contacts from a collision

- Implemented custom time of impact calculations.
  - We calculated an approximate time of impact by sweeping the motion using subdividing timesteps
  - Not the most efficient, but good enough for our needs

Fixing Deformable Shape Problems

- Our RasterShapes could be deformed by adding or removing blocks.
  - The raster grids were also resizable on the fly

- Deforming shapes can either invalidate current contacts or create new collisions
  - We added code to detect and destroy contacts that were invalid
  - New collisions were detected automatically in the next frame

Customizing Box2D without Changing their Source

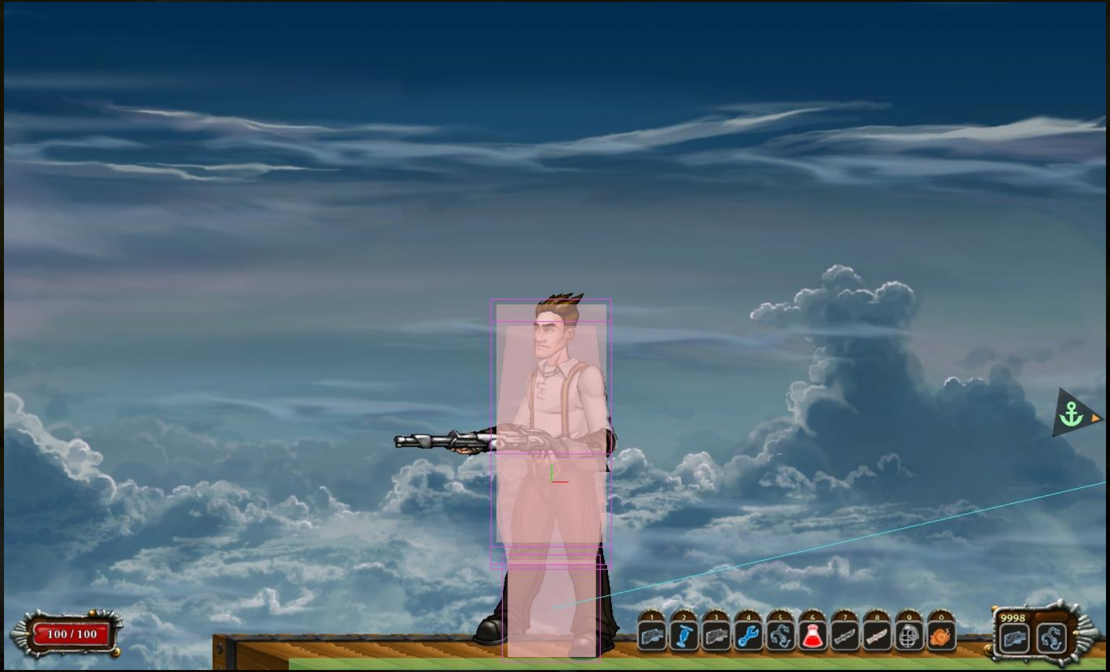
- Box2D lets you define a bunch of callbacks by extending their b2ContactListener class:
  - BeginContact
  - EndContact
  - PreSolve
  - PostSolve
- The most common use of these are to add logic associated with physics events.
- The PreSolve callback was especially useful since it allows you to modify or disable contacts before they are given to the solver.
  - This allowed us to implement many of the required features found in platforming games

Platforming and Ship Physics

Character Shape Setup

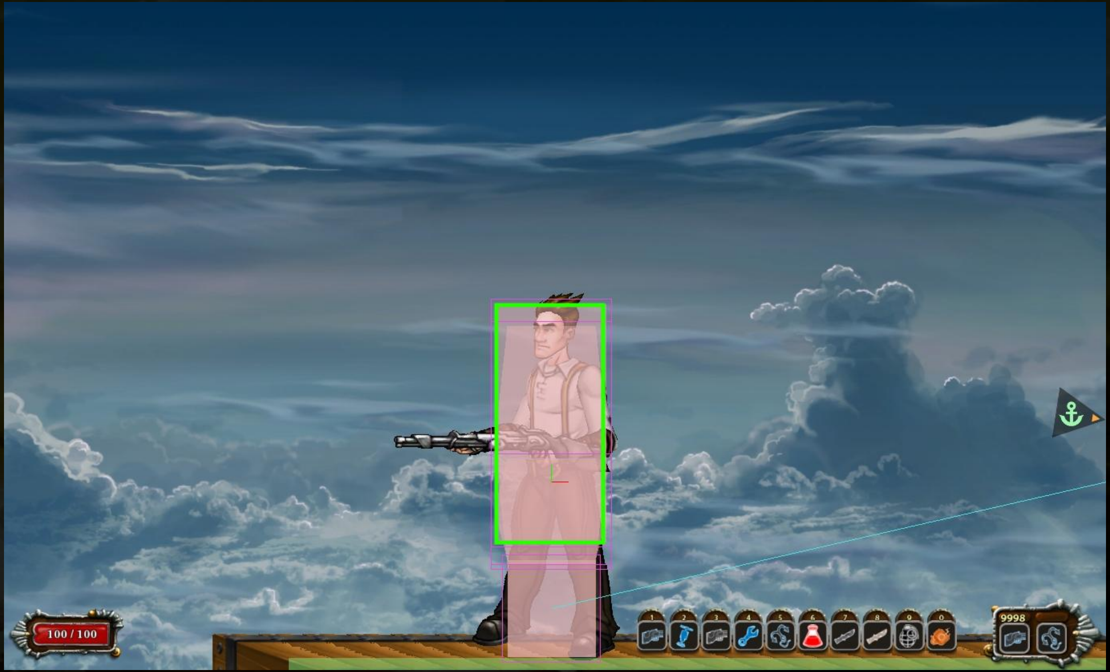
5 Different Shapes on characters:

- Stand up sensor
- Standing torso
- Standing legs
- Crouching torso
- Crouching legs



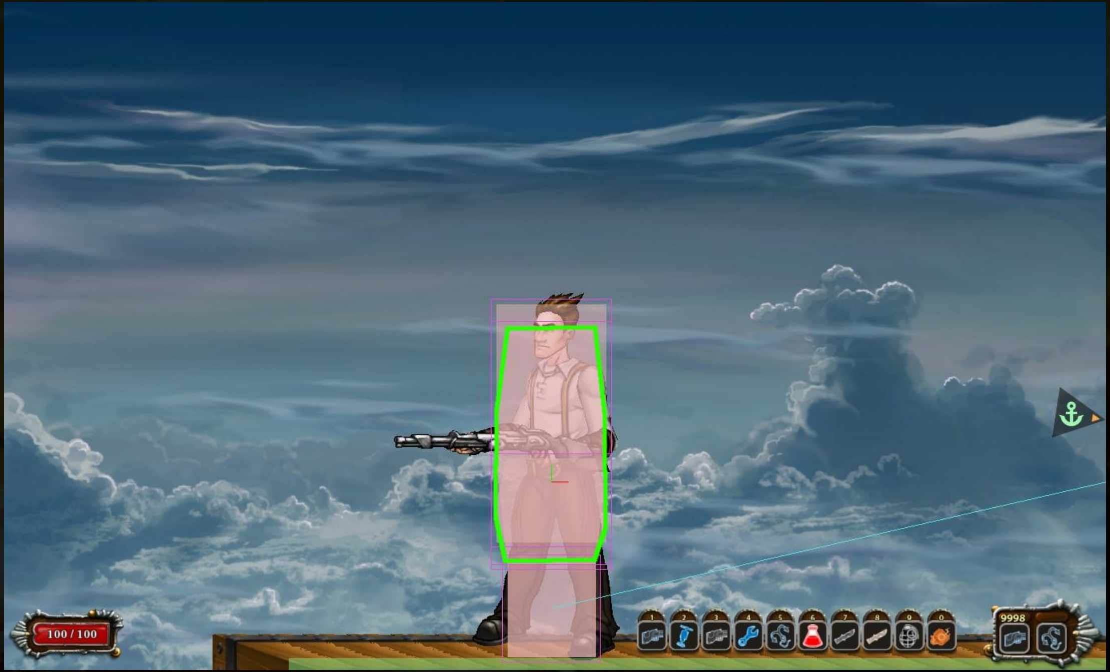
Character Shape Setup: Stand Up Sensor

- Used to detect obstructions that should prevent standing up.



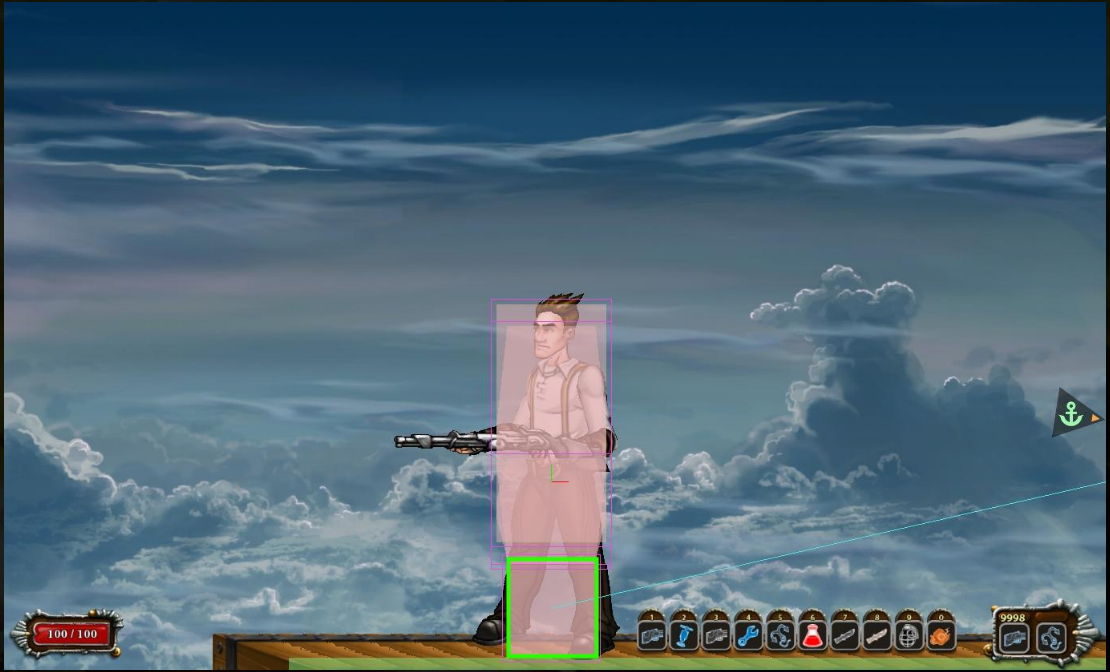
Character Shape Setup: Standing Torso

- Main character collision
- Top and bottom is slightly tapered to make it easier to jump and fall through tight spaces.



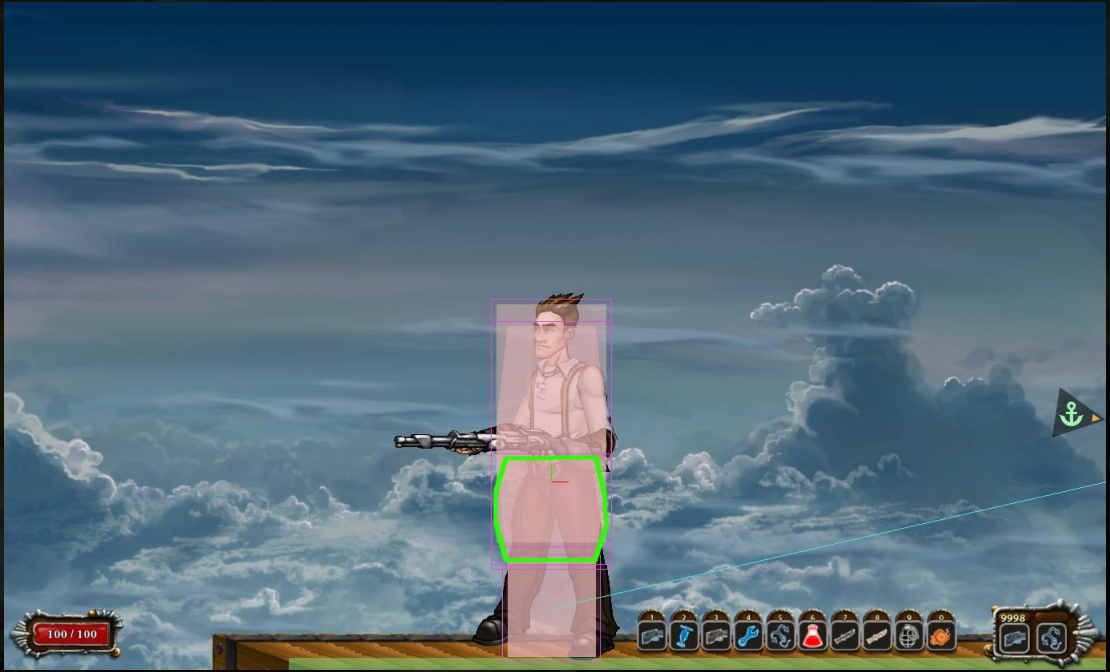
Character Shape Setup: Standing Legs

- Used to detect collisions that the character should step over.
- Slightly more than 2 blocks high.



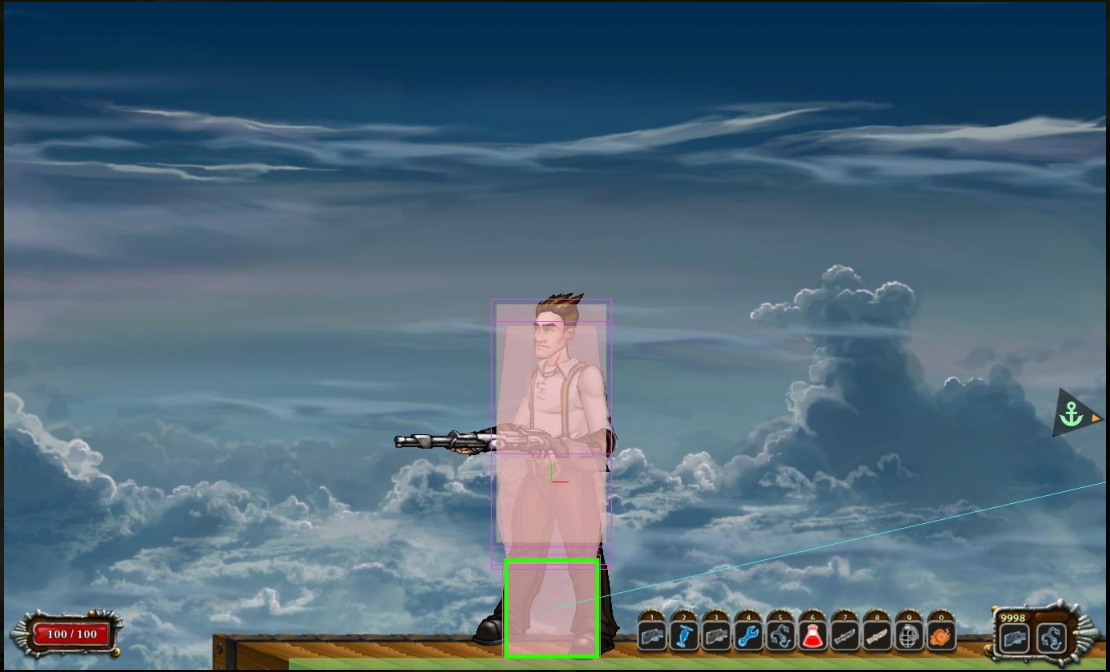
Character Shape Setup: Crouching Torso

- Similar to the standing torso.
- When crouching the standing collisions are disabled and vice versa.

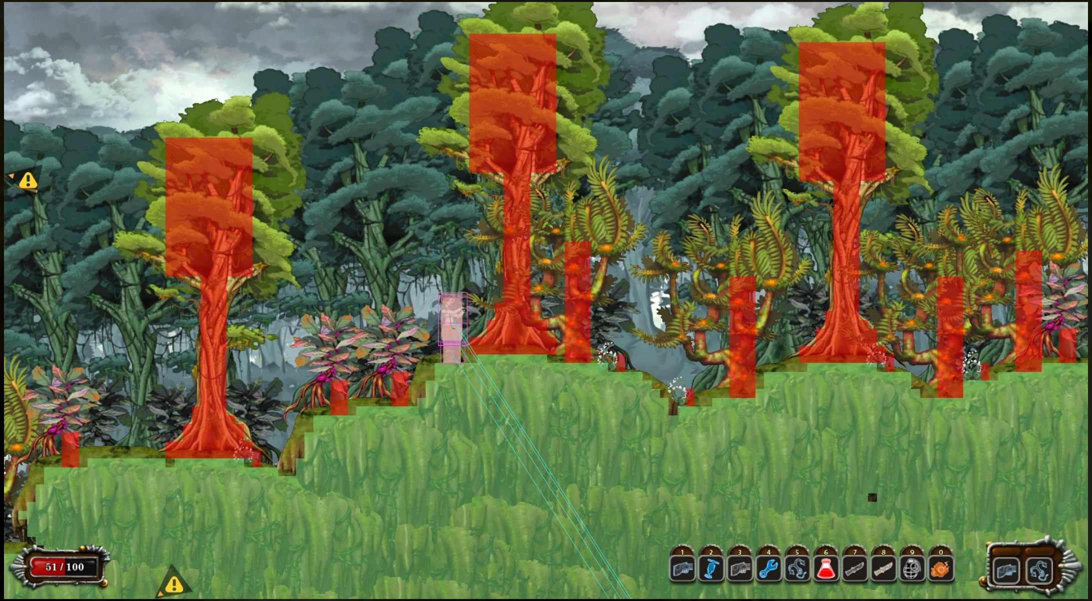


Character Shape Setup: Crouching Legs

- Similar to the standing legs.
- When crouching the standing collisions are disabled and vice versa.



Character Step-Up



- Stair-like surfaces, and irregular terrain is really common in Windforge.
- It would be really annoying to play without the auto step-up
- The step-up also made platforming more forgiving.

Step-Up Implementation

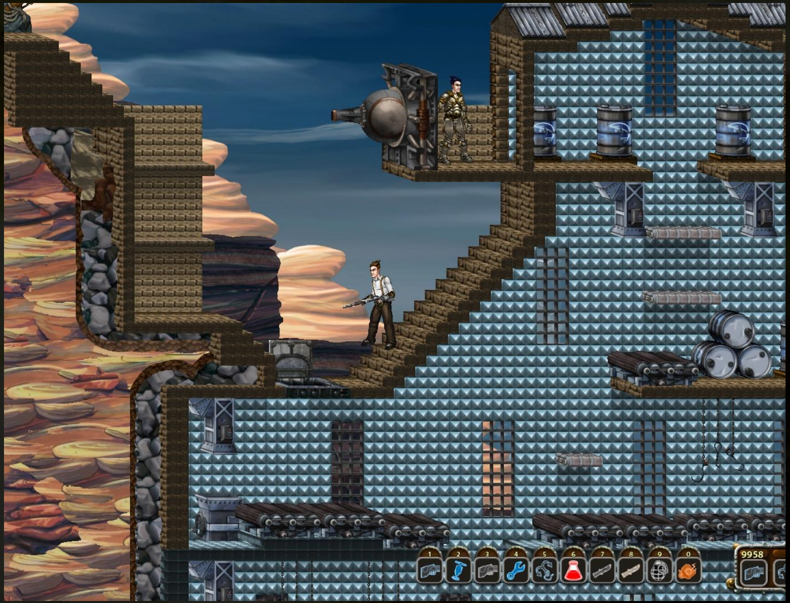
- Implemented step up by modifying the contacts in the PreSolve callback
- We got the solver to do the step-up for us by always making the normal point up
- To avoid occasional collision problems we also had to add extra checks to make sure it was safe to step up.

Step-Up Implementation

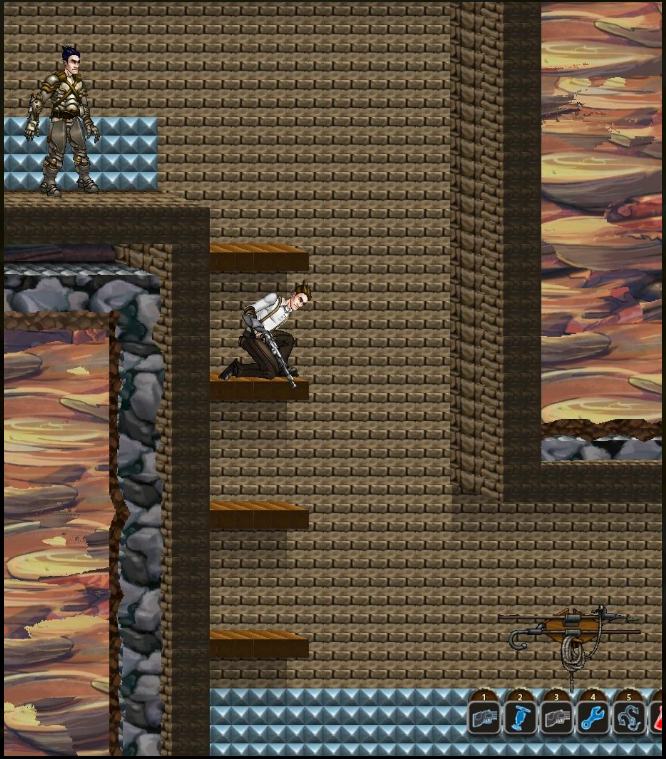
- This implementation worked well with the rest of the physics, and took very little effort.
- However, it would have been very hard to tune, if we wanted to customize the movement more.

Step-Up Implementation

- The step up behaviour sometimes made it difficult to fall through small holes.
- We ended up adding a special case to our implementation to disable the step-up if you are at the opening of a hatch.



One Way Platforms



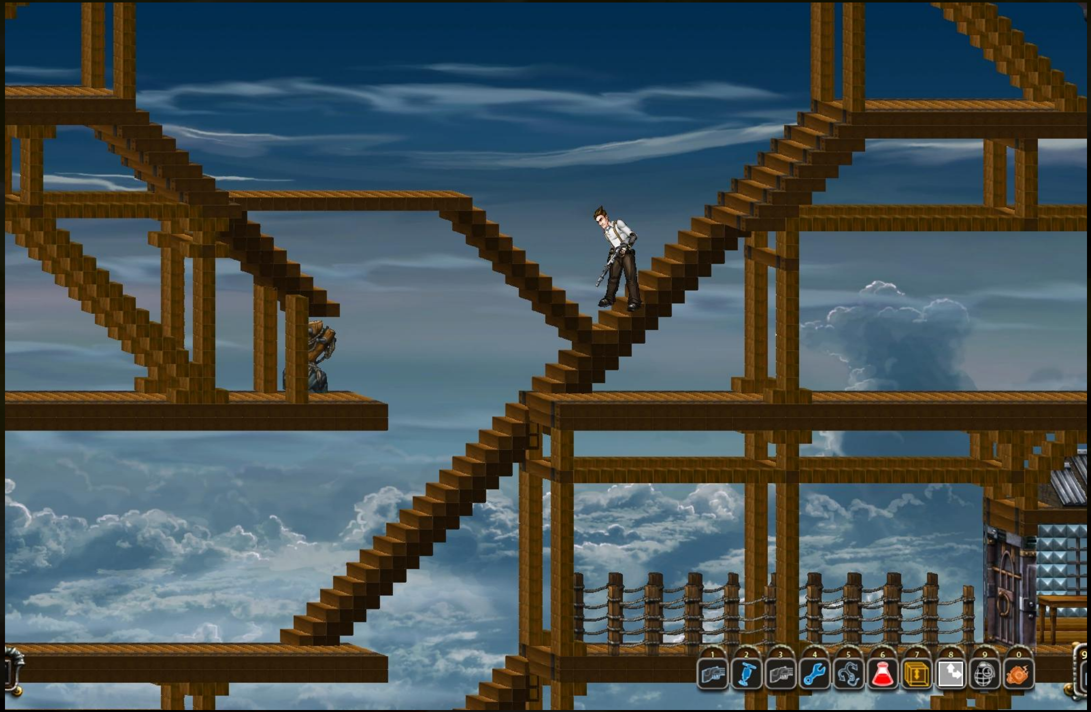
- We also implemented our one way platforms using the PreSolve callback

- Rules for colliding with one way platforms:
    - Torso shapes never collide with these
    - Ignore one way platform blocks without empty space in the grid above them
    - Ignore if you are moving upwards
    - Ignore if the platform is too high

One Way Platforms + Step-Up

- Building stairs out of one way platforms was really common.

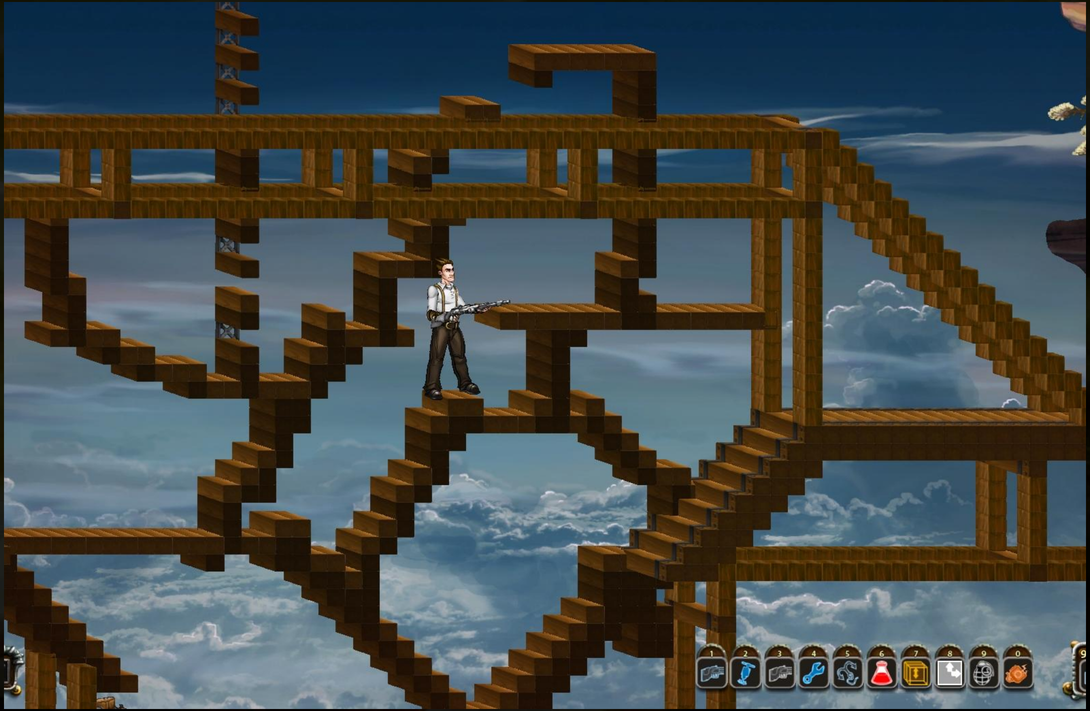
- There were a lot of extra cases we needed to handle to make it work well.



One Way Platforms + Step-Up

- Since the player can build things, we had no way to prevent non-ideal arrangements of blocks.

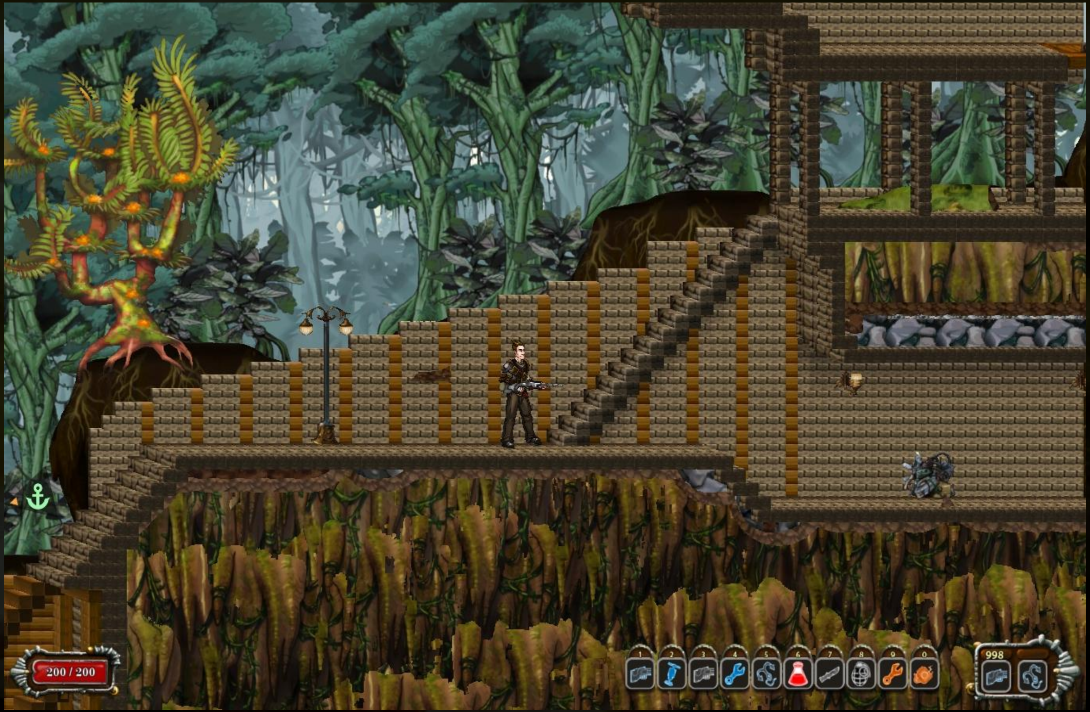
- Needed to make player movement as intuitive and non-frustrating as possible.



One Way Platforms + Step-Up

Some of the special cases we added:

- Don't do platform step up if you are standing on a normal block.
- Use very small step up distance for if you are in air
- Use normal step up distance if you are on platforms

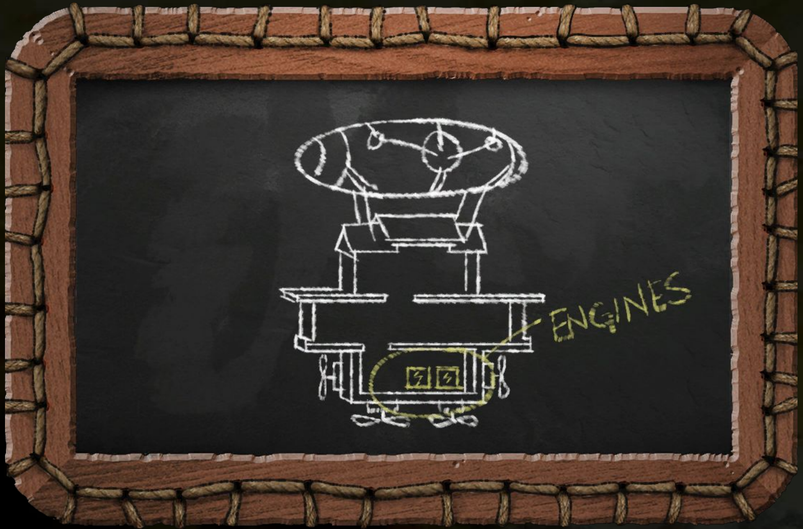


Ship Physics

- The ship movement is affected by how it is built

- Tried to keep the system as simple and easy to learn as possible, but keep it deep enough to make ship building interesting.

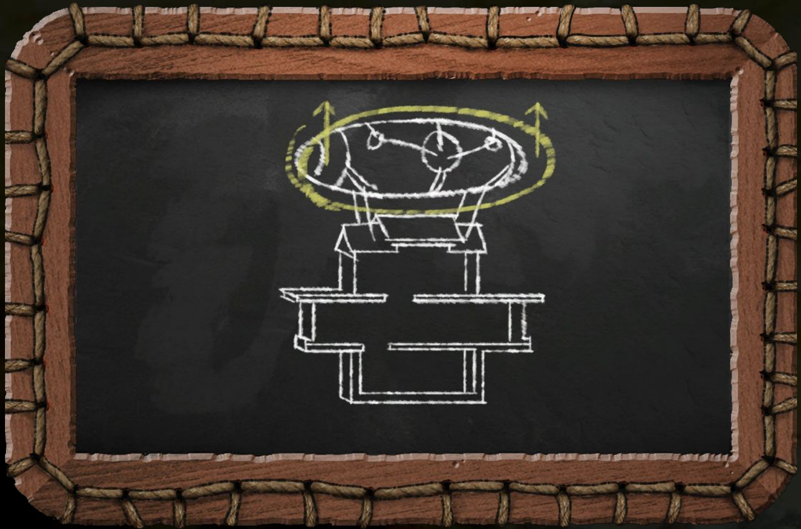
Ship Physics: Power Calculations



1. Calculate power load percentage
   - TotalAvailableEnergy / TotalEnergyRequired.

2. Reduce the effectiveness of components if load percentage is less than 100%
   - Propeller forces
   - Gun fire rate
   - Air purification rate
   - etc.

Ship Physics: Forces

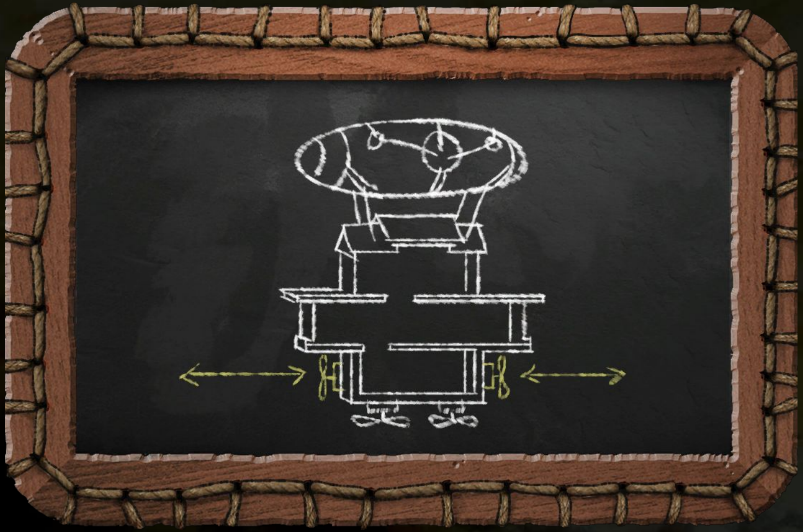


Movement affected by these forces:

- Current propeller thrust
- Max propeller thrust
- Force of gravity
- Buoyancy
- Wind

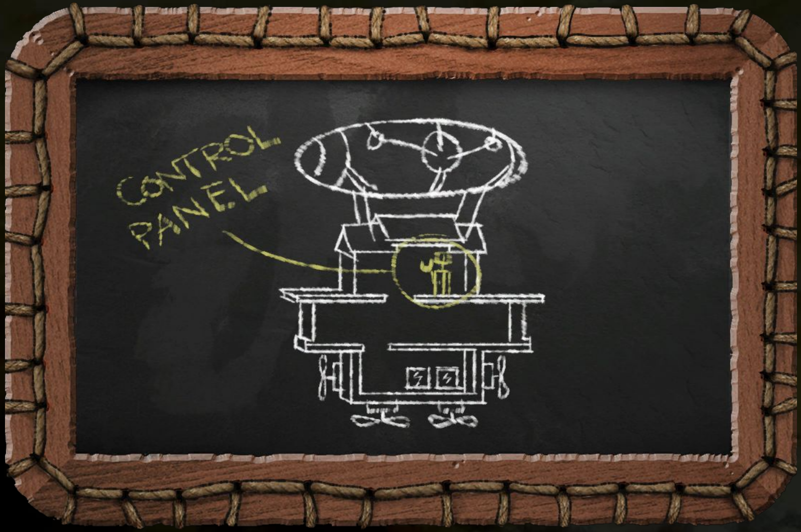
Mass is also calculated and used by the physics system.

Ship Physics: Propellor Thrust



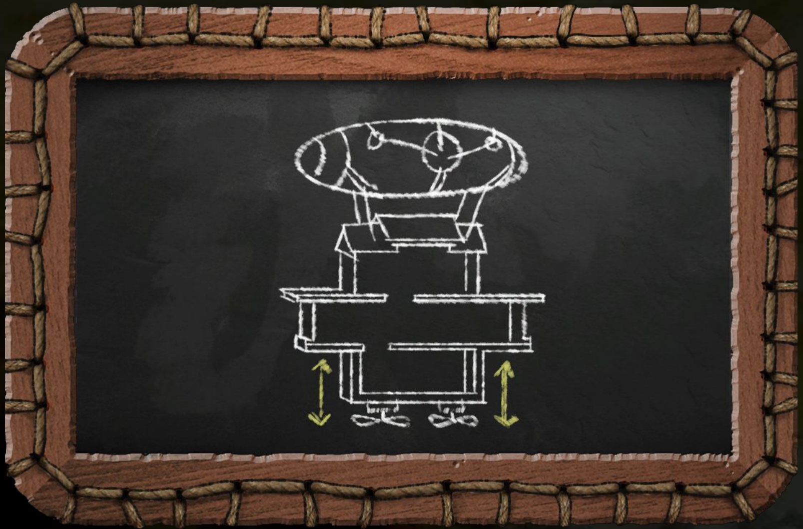
- Horizontal thrust = sum of horizontal propellers
- Vertical thrust = sum of vertical propellers
- Sampled with multiple rays to check if propellers were obstructed by ship walls, etc.
- The horizontal and vertical controls affected the percentage of thrust in each direction to apply.

Ship Physics: Controls



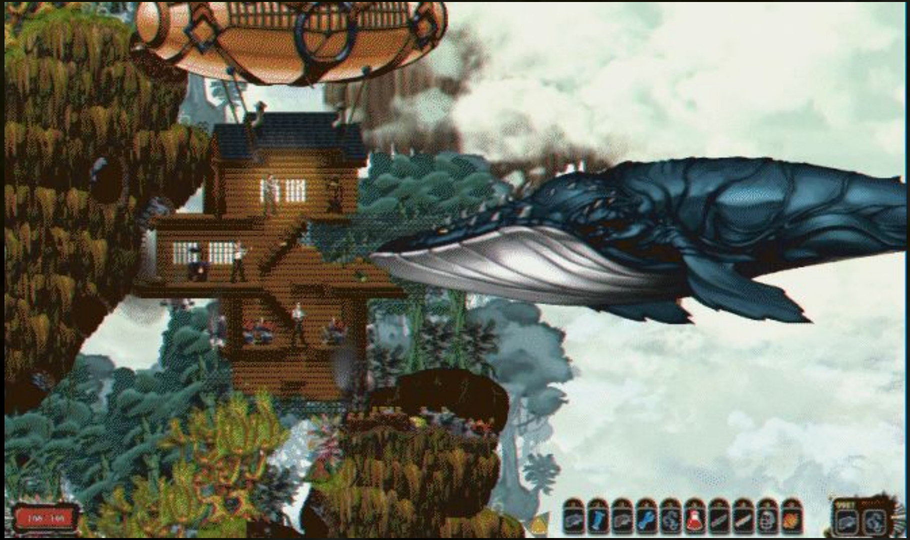
- Digital keyboard controls were too jarring. Faked analog controls using accelerations
  - Note: changing directions was instant
- Calculated the input needed to hold the ship stationary against wind and gravity.
  - “Centered” the input using these values.

Ship Physics: Fixing Ship Drift



- Character, and creature collision resolution sometimes pushed the ships down slightly.
  - Added an extra “fudge” force to correct for this.
- Various minute forces could cause ships to slowly drift away
  - If on screen we would try to push the ship back into place.
  - If off screen we would turn off physics as long as it was possible to fly.

Ship Physics: Ship Collisions

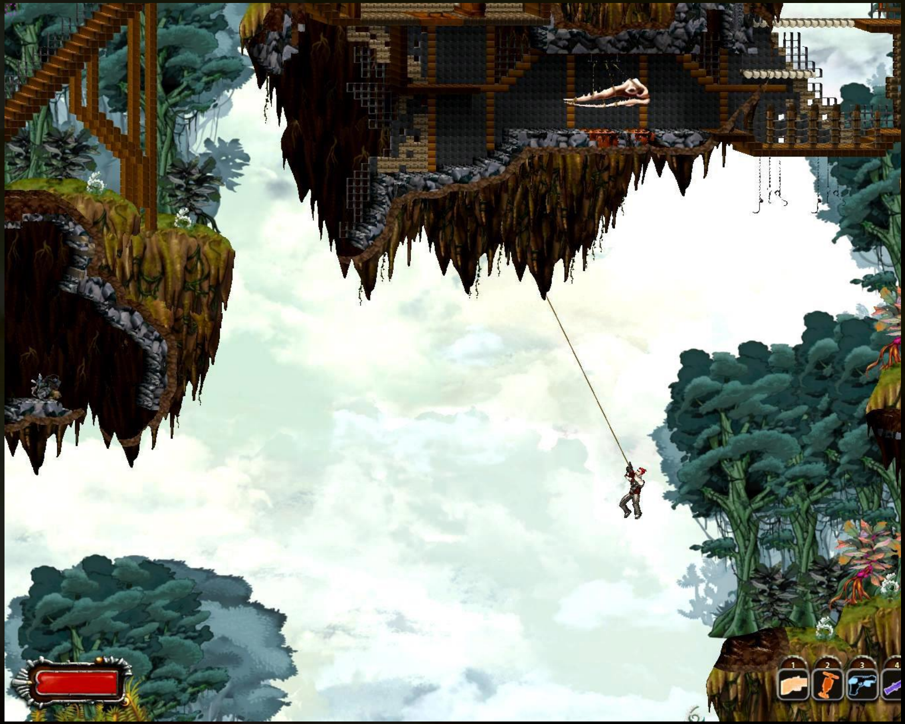


- Collision resolution was handled by Box2D
- Collision damage was calculated using the impulses from the PostSolve callback
  - Queued this info up and applied this after the physics step to avoid modifying shapes while solving
- Distributed impulses evenly among affected blocks
  - “Sharp” block arrangements concentrate damage.

Grappling Hooks

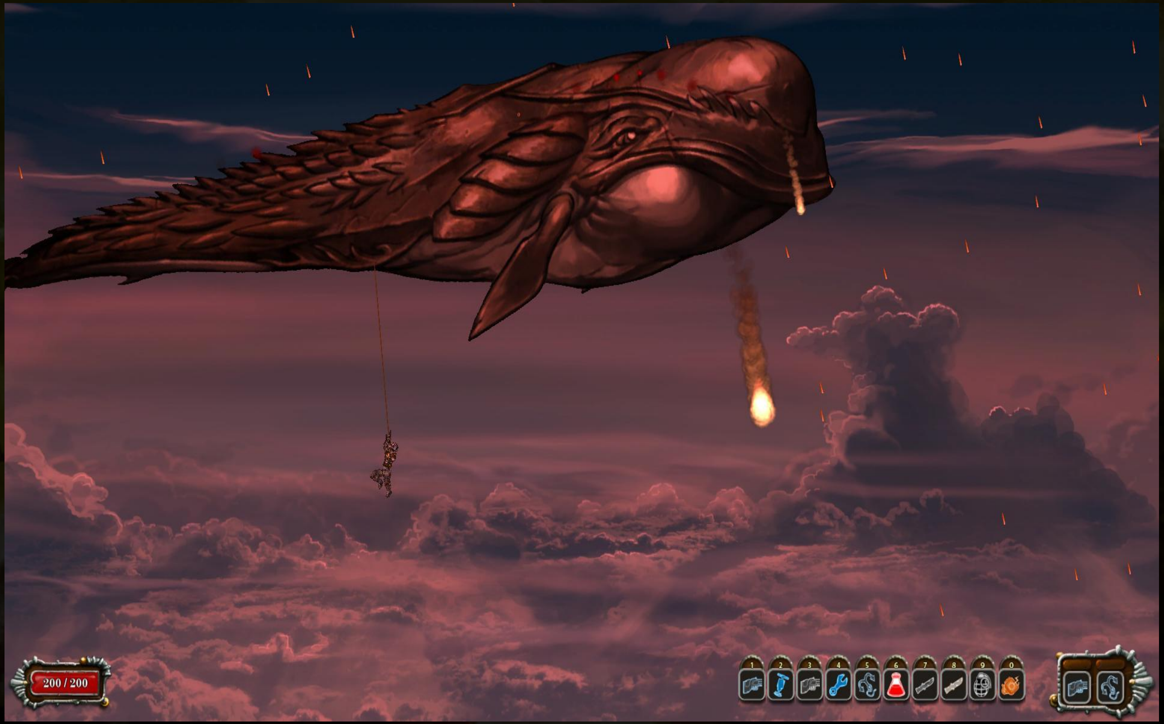
Grapple Hooks

- Windforge had a lot vertical space to explore that was difficult or impossible to jump.
- This was difficult to predict and control.
- The grappling hook was an essential tool for exploring the world.



Design Goals for the Grappling Hook

- Responsive
- Predictable
- Intuitive
- Easy to Control
- Minimal Special Cases

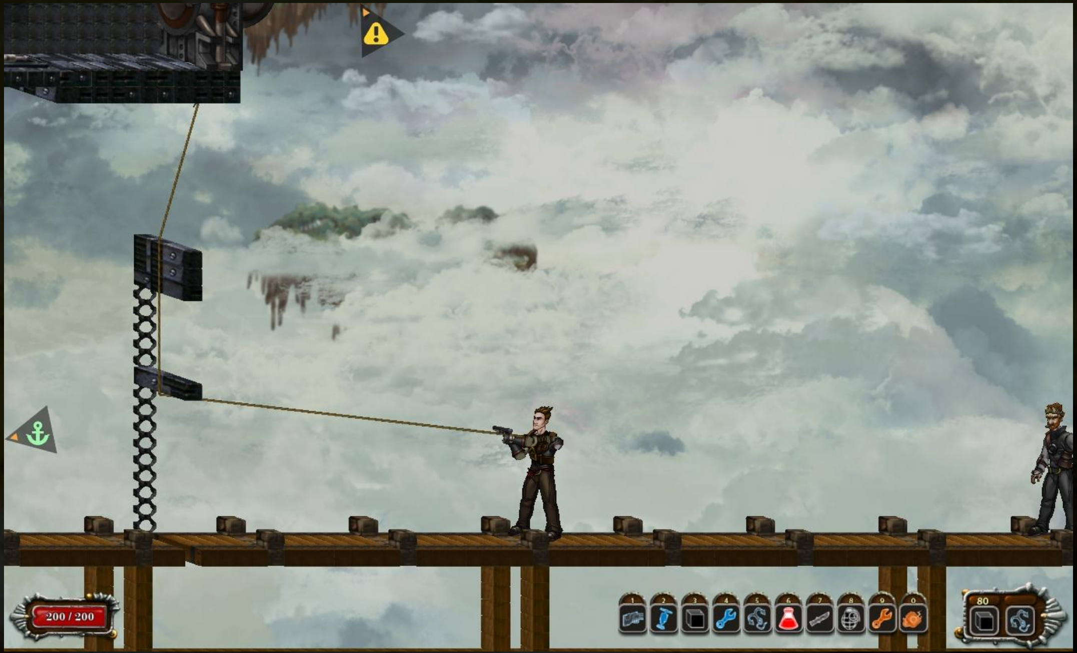


Realism was not a priority for the grappling hook and was sometimes sacrificed.

Anatomy of A Grappling Hook

The main grappling hook classes

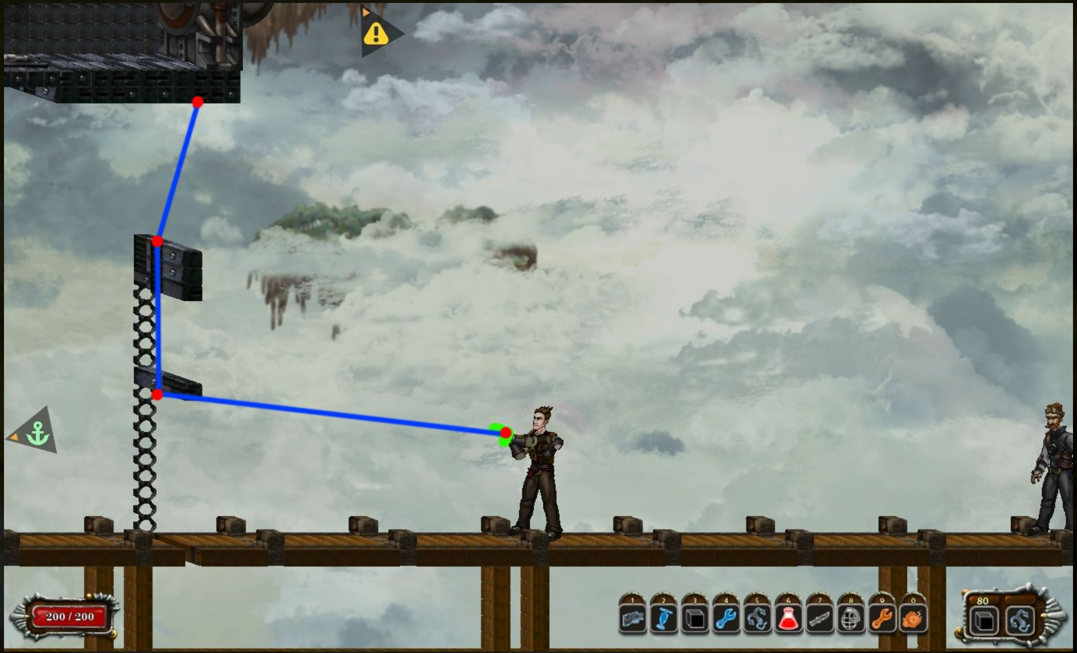
- Grappling hook item
- Cable Segment
- Cable Node



Anatomy of A Grappling Hook

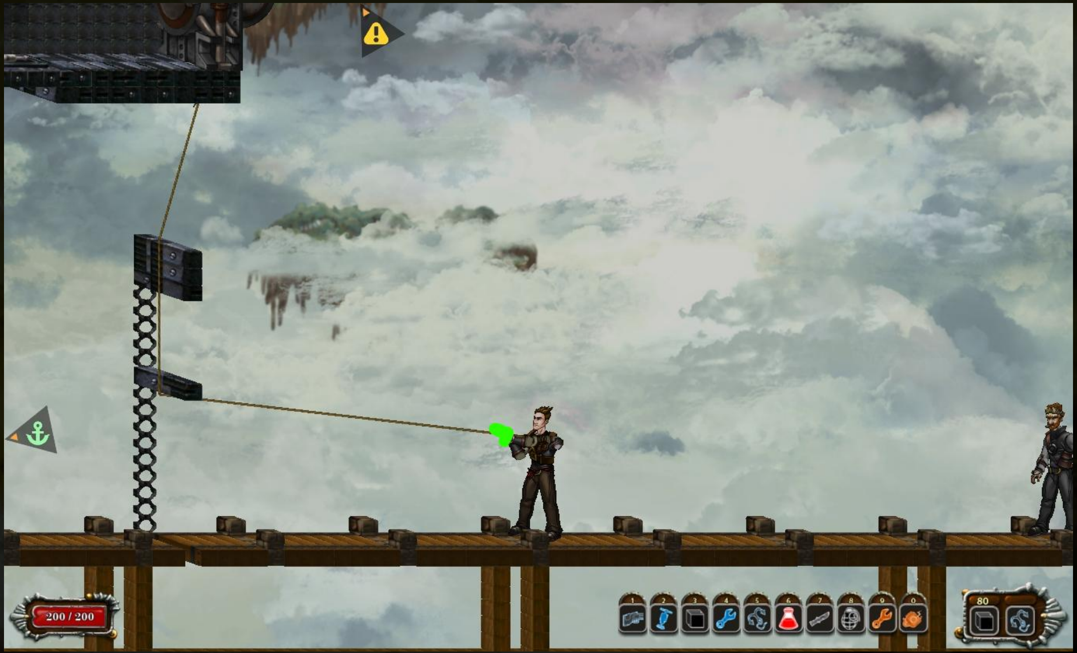
The main grappling hook classes

- Grappling hook item
- Cable Segment
- Cable Node



Grappling Hook Item

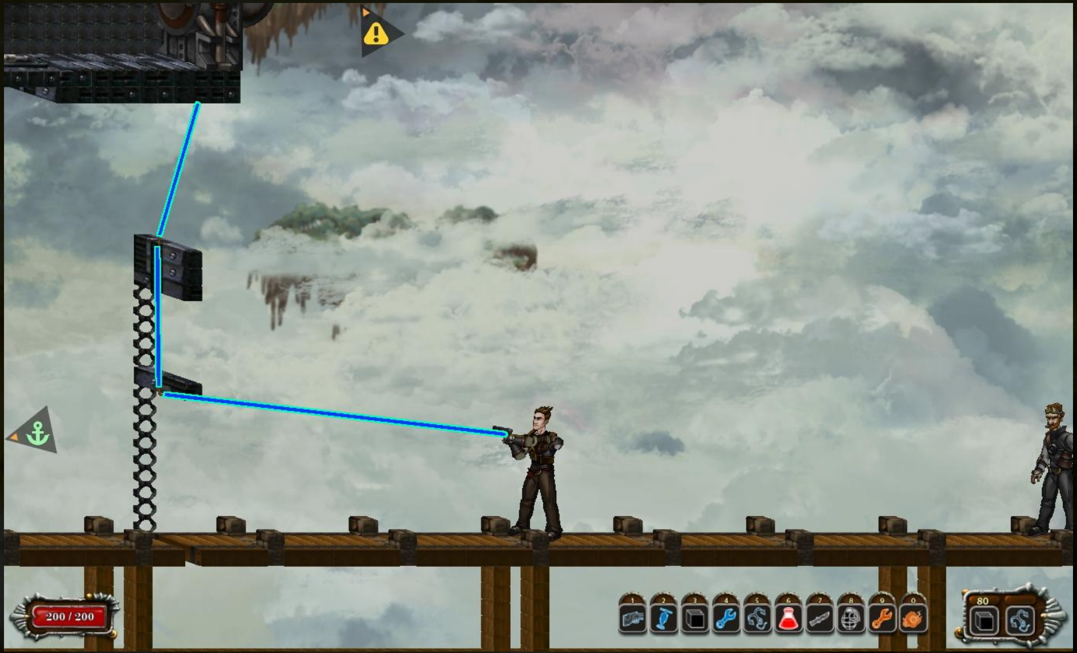
- Handles the following:
    - Grappling hook physics
    - Grappling hook state machine
    - Manages the nodes and segments
    - Splitting or combining segments
    - Sounds



Cable Segments

Mainly responsible for:

- Cable visuals
- Connecting to nodes



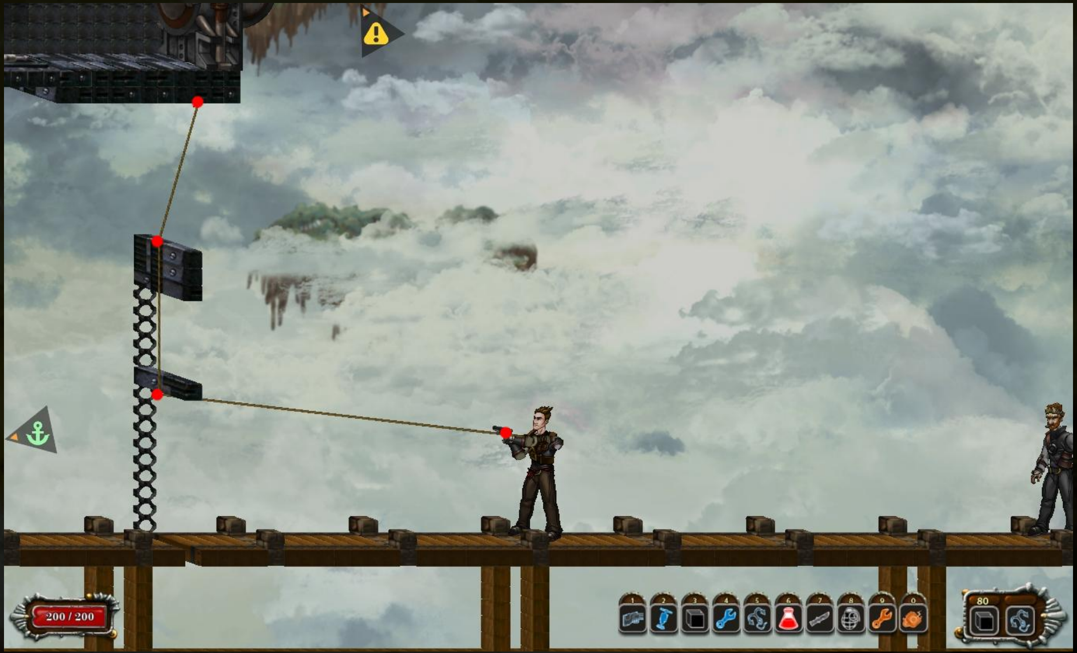
Cable Nodes

Mainly responsible for:

- Connecting the segments
- Anchoring to objects
- Tracking the *bend sign*

The node at the item is the *Anchor Node*

The node that sticks to stuff is the *Latch Node*



Grappling Hook Main States

- Ready To Fire
- Ballistic
- Latched

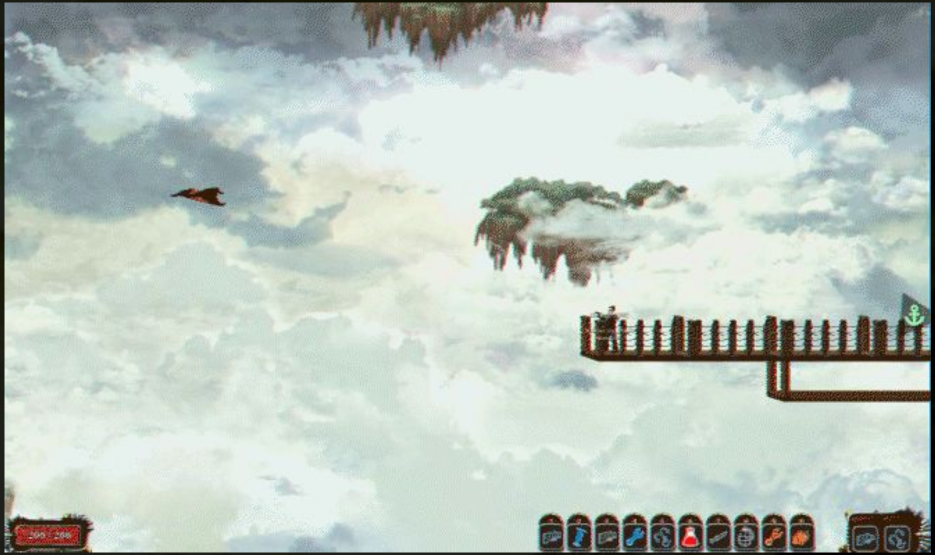
Ballistic State

- When the grappling hook is fired it will go into the ballistic state
- The latch node will travel using a simple ballistic arc
  - If the ballistic node distance from the anchor exceeds the max range:
    - The position will be clamped
    - The velocity along the cable direction will be set to zero

- When the latch node hits something, the grappling hook will stick to it and go to its latched state.

Ballistic State: Collision Detection

- Used a ray cast from the anchor node to the latch node.
- Not *realistic* but it was simple and robust
- Made aiming the grappling hook really forgiving



Latched State

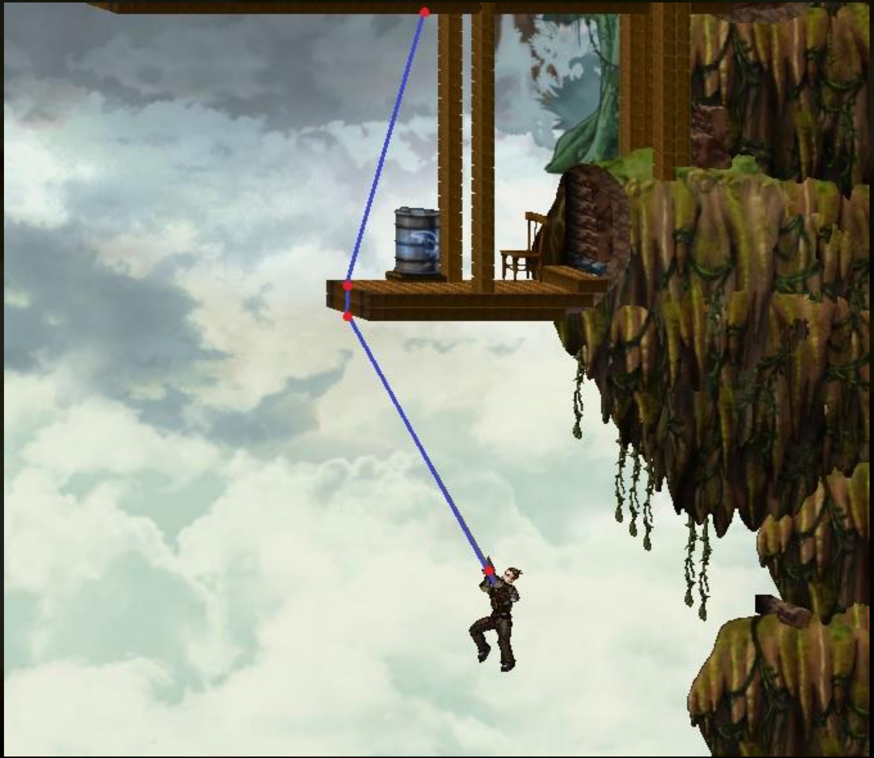
Responsible for behaviour when latched to something:

In particular:

- Custom spring forces to make it feel like a rope
- Controls for swinging and changing length of cable
- Collision detection to check if cable should bend
- Checks if cable should un-bend

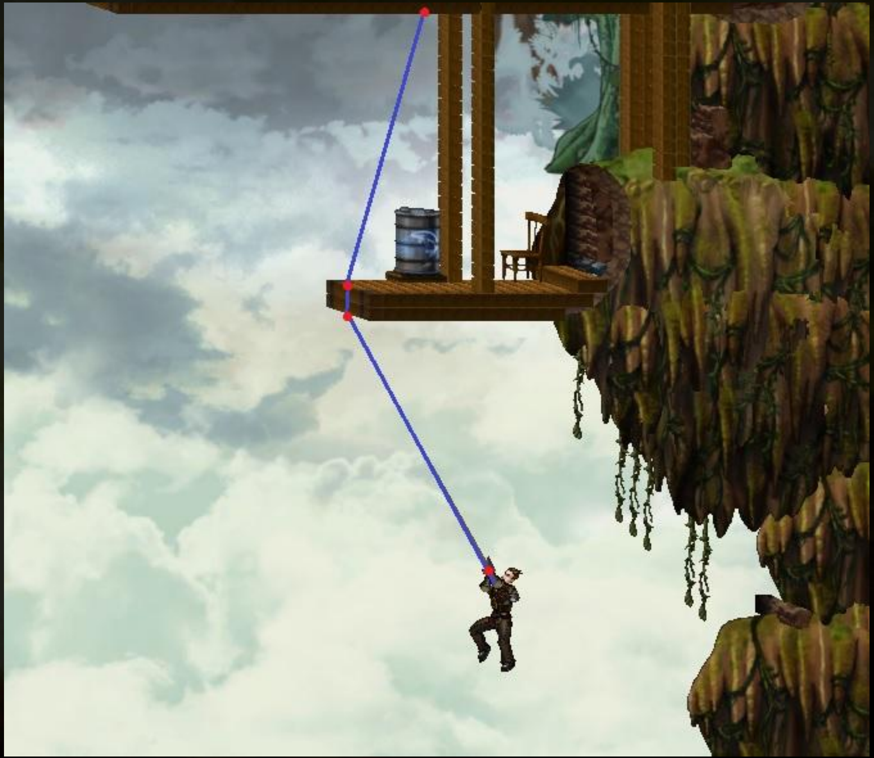
Latched State: Spring Forces

- Calculate the length of the entire cable: *CurrentLength*
- Character controls the rest length: *DesiredLength*
- Spring direction is between character and first attached node.



Latched State: Spring Forces

- Used simple damped spring formula
- Spring force based on the ratio of the rest length difference over the max length
- Damping force based on the velocity of character in the direction of the spring



Latched State: Spring Forces

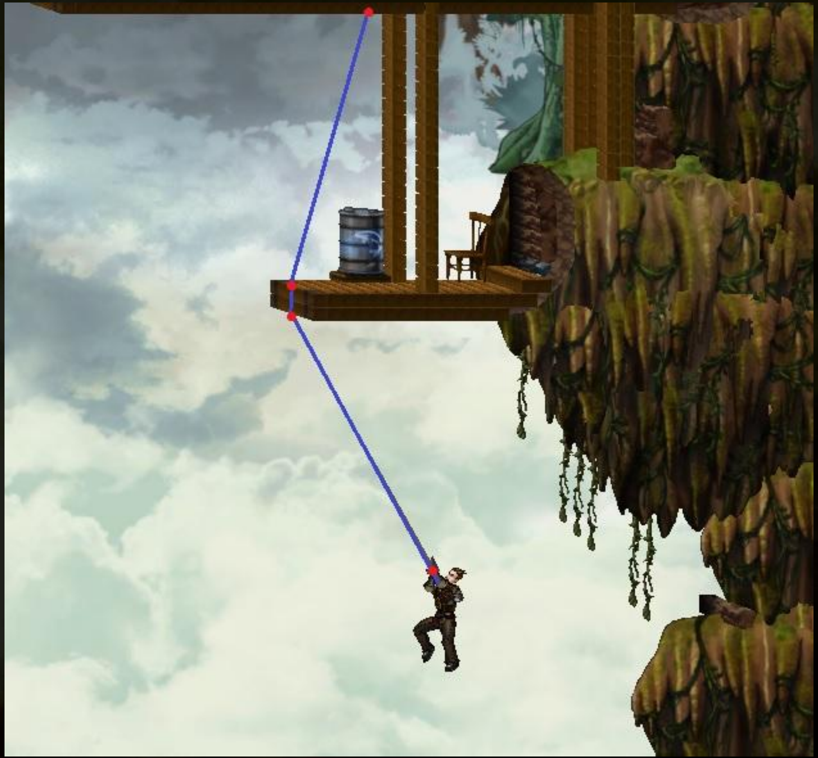
FinalForce = Max(0, SpringForce - DampingForce)

SpringForce =

MaxForce * (CurrentLength - DesiredLength) / MaxLength

DampingForce = Damping * SpringDir · PlayerVel

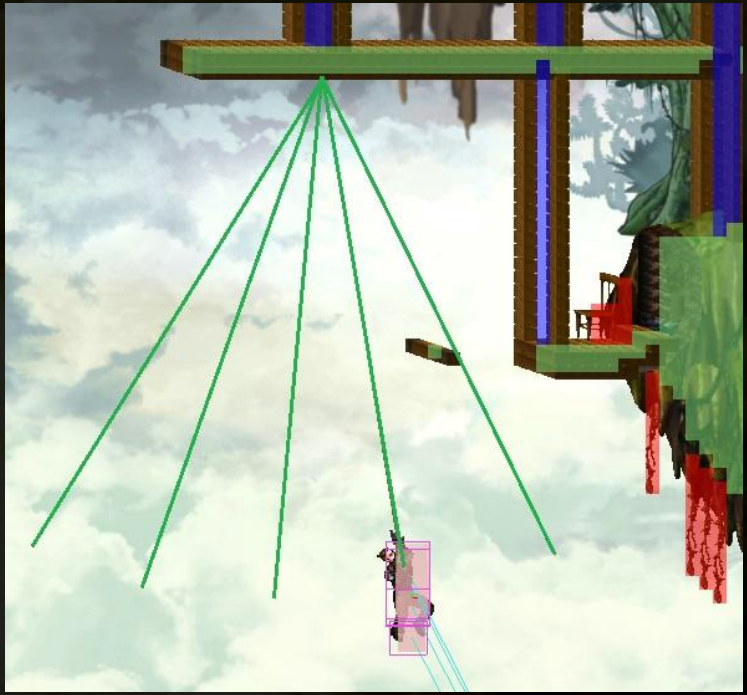
(SpringDir is the direction of the first segment)



Latched State: Collision Detection

- Used raycasts between the nodes of each segment.

- Had to do some extra work to deal with per-frame sampling inaccuracies



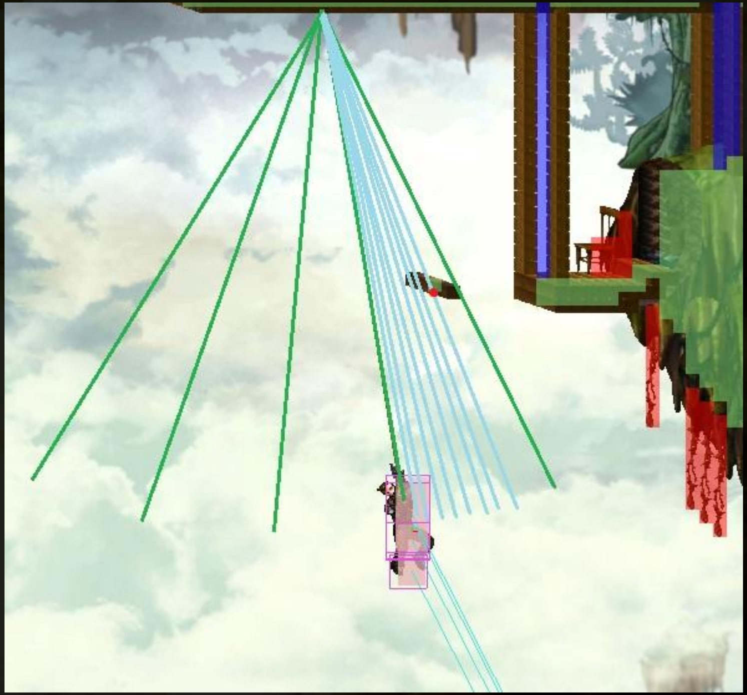
Exaggerated per frame segment positions

Latched State: Collision Detection

- Nodes tracked their previous positions

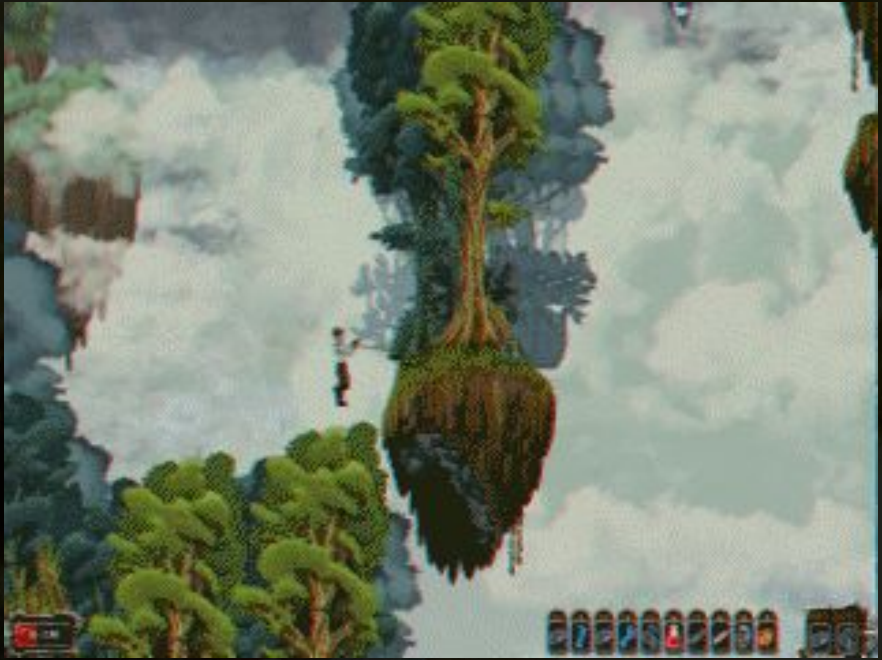
- Added extra ray casts in a sweeping fashion from the previous positions to current

- After position calculate block bend corner
  - Used closest open corner to the previous position of the segment



Exaggerated per frame segment positions + extra sweeping samples

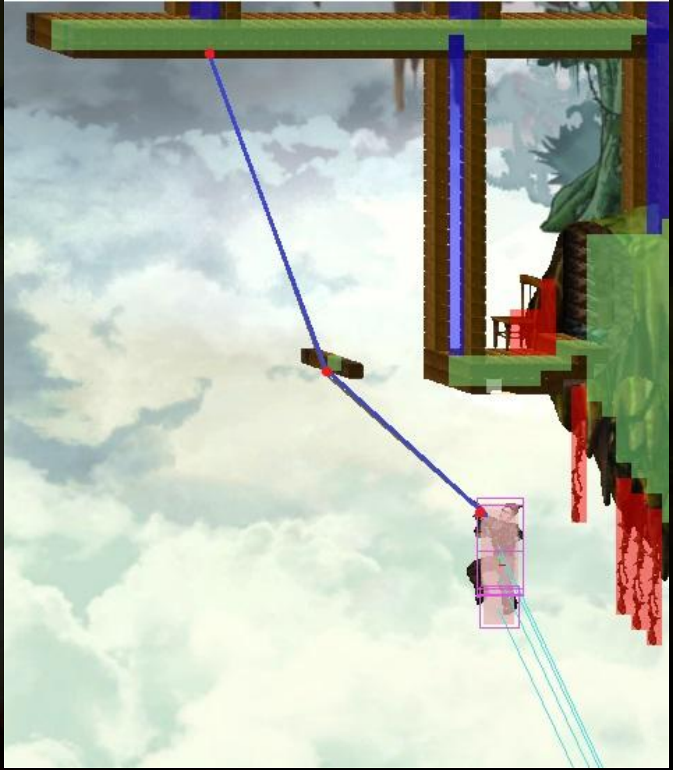
Latched State: Bendy Cables



Making the cables bend around corners greatly improved the look and feel of the grappling hook.

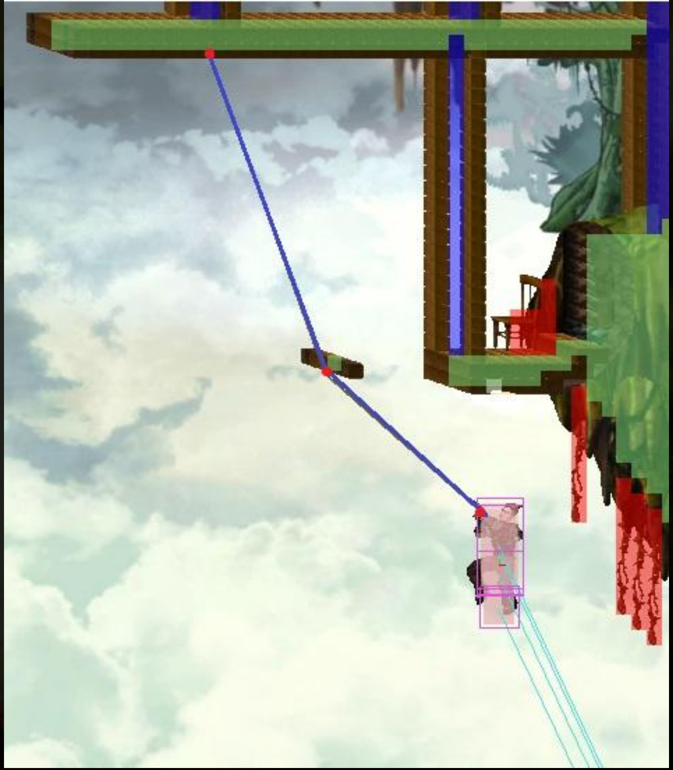
Latched State: Bending Cables

- Split the cable when you detect collisions
- Insert the new splitting node at the bend corner you calculated.
- Calculate and store the *bend sign* at the splitting node
  - The bend sign is the sign of the 2D determinant of the segment directions from splitting node.



Latched State: Unbending Cables

- Recalculate the bend sign each frame on bending nodes
- If the bend sign is different than the original then you can unbend the cable
    - Remove bend node and combine neighbour segments



Latched State: Overstretching

Cable could sometimes get overstretched:

- Commonly caused attaching to moving ships and being obstructed by geometry.
- A minor amount of overstretching was acceptable, but excessive overstretching sometimes caused explosive problems
- Fixed using a combination of max forces, max lengths, and unlatching the cable in extreme cases.

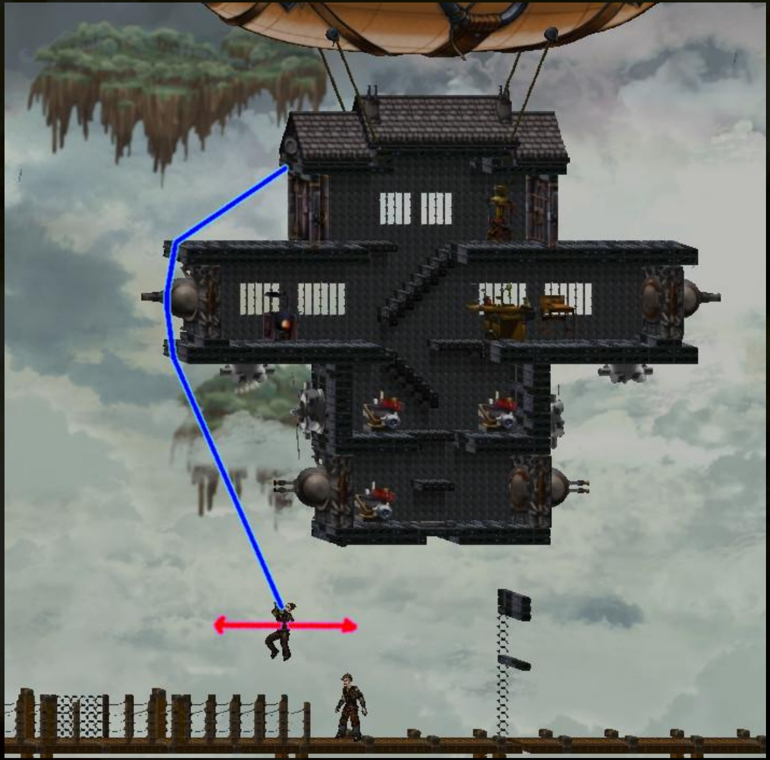
Control Improvements

Tweaking little details in the controls made a big difference:

- In-air control forces
- Wall push off impulse
- Cable jump
- Auto lengthen cable while walking

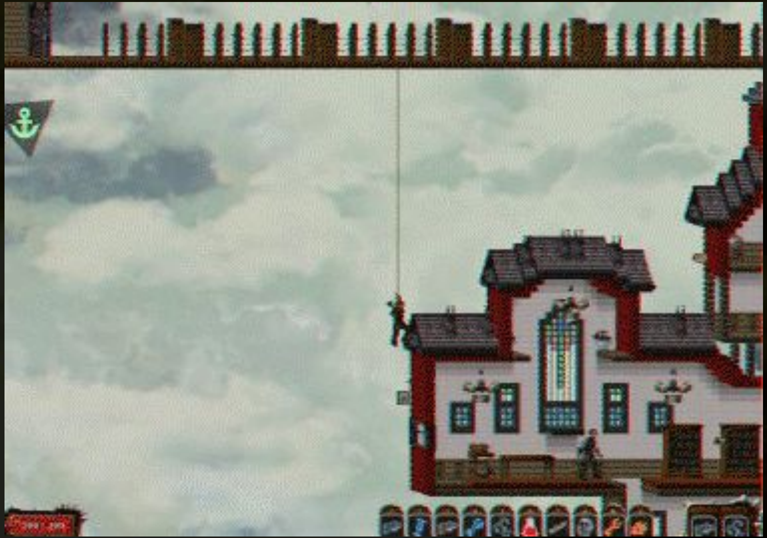
In-Air Control Forces

- Control swinging by adding left and right forces that correspond to the controls
- Clamp force if it would make you exceed a max speed.
- Use 0 linear damping if there is input and a constant linear damping if no input
  - Extra responsive if there is input
  - Makes player come to rest eventually if no input



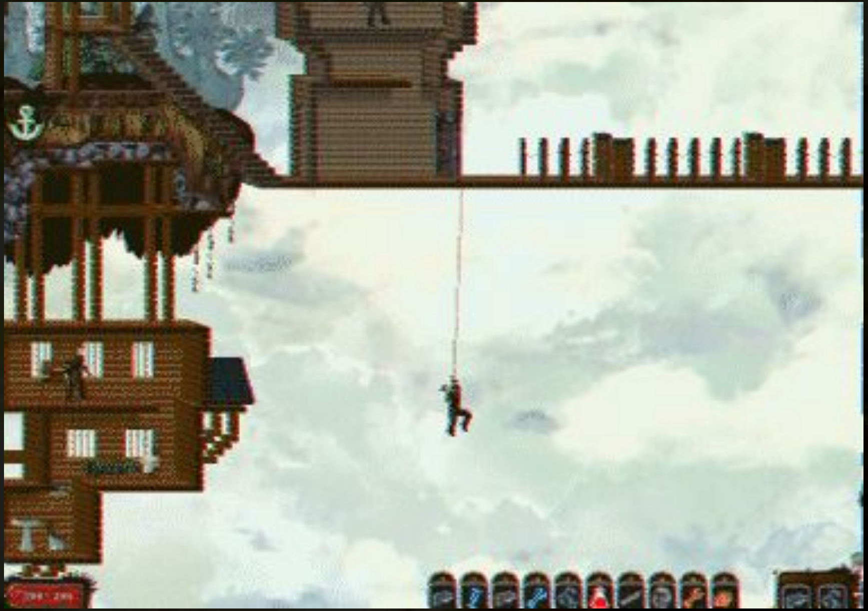
Wall Push Off Impulse

- Inspired by rappelling pushing motion
- Apply an extra impulse if you are touching a wall when pressing left or right.
- Gives an extra boost to get away from the wall
  - The in air forces aren’t enough



Cable Jump

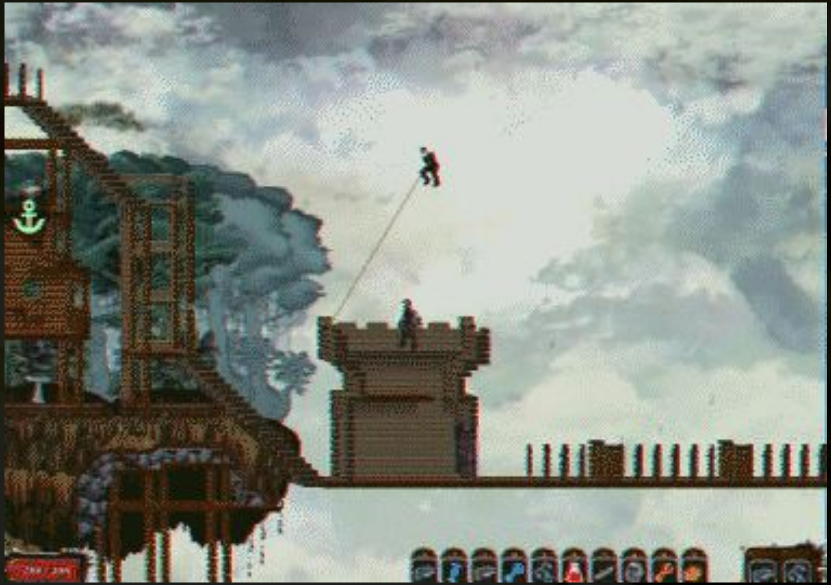
- Allow a small in-air jump if the cable is latched to something
- Gives the player an extra boost to help get on top of platforms, mid-air grappling, etc.
- Makes it much easier to climb along walls and ceilings



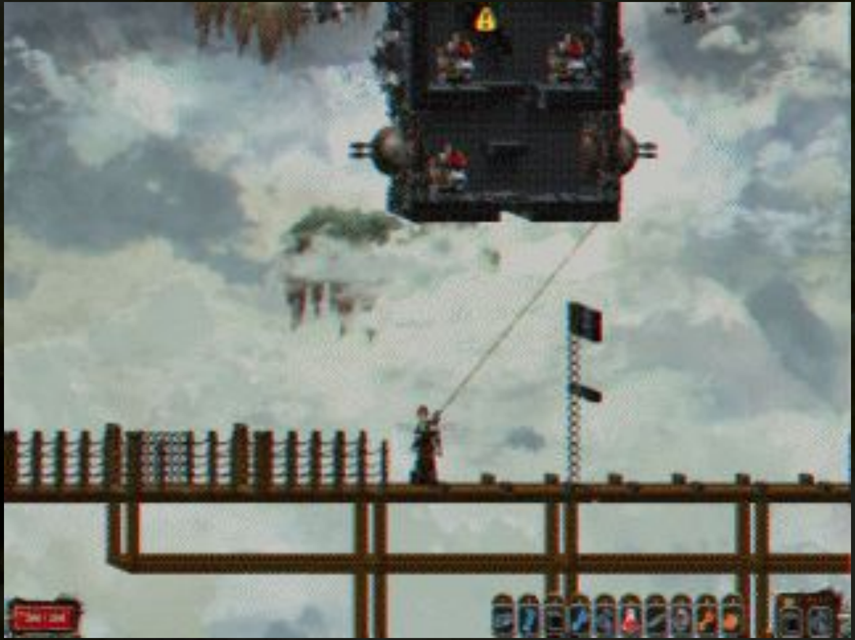
Cable Jump

Advanced players can take advantage of this:

- Double jump during fights
- Break falls to protect against falling damage
- Quickly traverse terrain over any surface

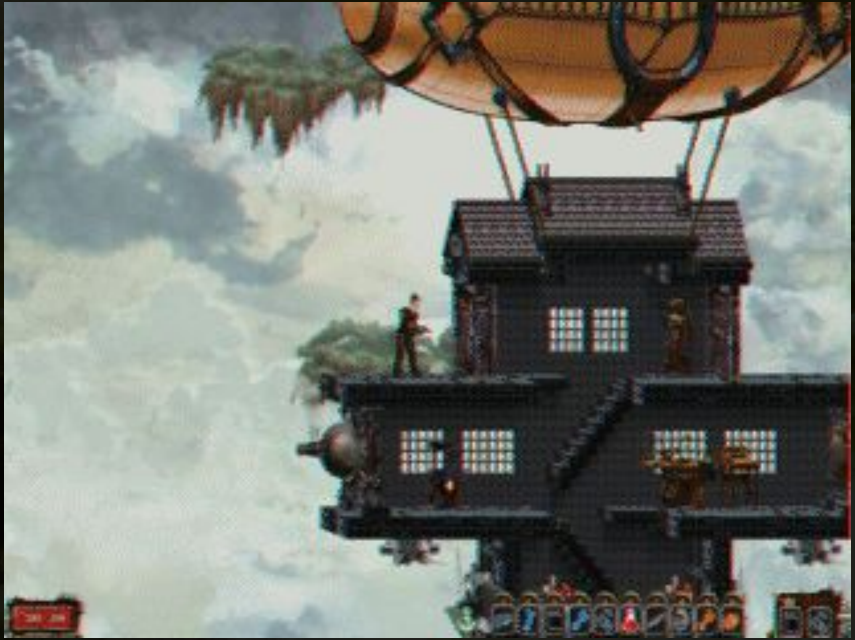


Auto Lengthen Cable While Walking



Walking on the ground felt awkward until we added auto lengthening / shortening while walking.

Auto Lengthen Cable While Walking



This feature also made exploring easier since you could latch on to the ground and lower yourself down

Extending Grappling Hook to 3D

- The same concepts used for this grappling hook apply to 3D.
    - (My first implementation of this was actually for an unreleased first person shooter)

- Bending around corners becomes a lot more complicated in 3D
    - Slightly more complicated math is needed to handle bending / unbending
    - To make things feel right you'll want to allow the cable to slide along edges (line segments)

Random Thoughts

- There is still room for improvement and polish with our physics
  - Additional polish
  - Performance improvements
  - (and a few lingering bugs...)
- Dealing with subtle details and tuning often took longer than the base features.
- Box2D worked out really well for us.
  - Easy to use
  - Free and open source
  - Straightforward to modify
  - It was also the easiest physics engine I've worked with when implementing one way platforms.

Questions?

Contact Info:

Evan Hahn

evan.hahn@snowedin.ca

@ehahnda